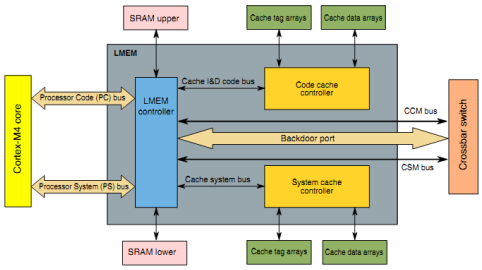
Section 2. Memory Architecture

Memory Organizations and Cache

Outline

- Local Memory Controller (CACHE)
- Memories and Memory Interfaces

Local Memory Controller



Memory map/Register Definition

LMEM memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
E008_2000	Cache control register (LMEM_PCCCR)	32	R/W	0000_0000h	28.2.1/ 697
E008_2004	Cache line control register (LMEM_PCCLCR)	32	R/W	0000_0000h	28.2.2/ 698
E008_2008	Cache search address register (LMEM_PCCSAR)	32	R/W	0000_0000h	28.2.3/ 700
E008_200C	Cache read/write value register (LMEM_PCCCVR)	32	R/W	0000_0000h	28.2.4/ 701
E008_2020	Cache regions mode register (LMEM_PCCRMR)	32	R/W	AA0F_A000h	28.2.5/ 702
E008_2800	Cache control register (LMEM_PSCCR)	32	R/W	0000_0000h	28.2.6/ 705
E008_2804	Cache line control register (LMEM_PSCLCR)	32	R/W	0000_0000h	28.2.7/ 706
E008_2808	Cache search address register (LMEM_PSCSAR)	32	R/W	0000_0000h	28.2.8/ 709
E008_280C	Cache read/write value register (LMEM_PSCCVR)	32	R/W	0000_0000h	28.2.9/ 710
E008_2820	Cache regions mode register (LMEM_PSCRMR)	32	R/W	AA0F_A000h	28.2.10/ 710

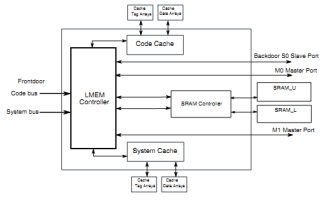
See K70 Reference manual P696 for details

LMEM Function

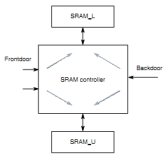
The Local Memory Controller receives the following requests:

- Core master bus requests on the Processor Code (PC) bus
  - Processor Code accesses are routed to the SRAM_L if they are mapped to that space. Allother PC accesses are routed to the Code Cache Memory Controller.
- Core master bus requests on the Processor Space (PS) bus
  - Processor Space accesses are routed to the SRAM_U if they are mapped to that space. All other PS accesses are routed to the PS Cache Memory Controller.
- SRAM controller requests from all other bus masters on the backdoor port.
  - All LMEM backdoor port accesses are for the SRAM controller. These accesses go to the SRAM_L or the SRAM_U depending on their specific address.

SRAM Function



SRAM Configuration



SRAM access diagram

Cache Function

Both caches have a 2-way set-associative cache structure with a total size of 8 KBytes. The caches have 32-bit address and data paths and a 16-byte line size. The cache tags and data storage use single-port synchronous RAMs.

- TAG function uses two 256 x 22-bit RAM arrays
- DATA function uses two 1024 x 32-bit RAM arrays.

CACHE - 8 KByte size = (256 sets) x (16-byte lines) x (2-way set-associative)

TAG:

- address[31:12] used in tag for compare (hit) logic
- address[11:4] used to select 1 of 256 sets
- address[3:0] not used

DATA

- address[31:12] not used
- address[11:4] used to select one of 256 sets
- address[3:2] used to select one of four 32-bit words within a set
- address[1:0] used to select the byte within the 32-bit word

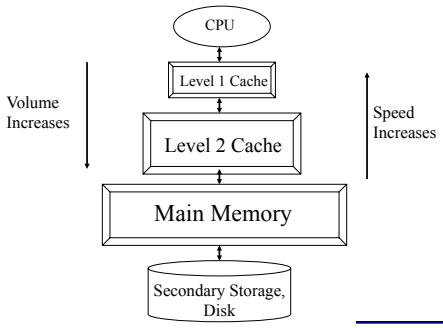
Memory System Architecture

- Concepts of memory hierarchy
  - Quantitative principles of computer design
    - Smaller is faster
    - Amdahl's Law:

      If we make an enhancement on a part of a computer, the overall performance gain is limited by the fraction of time when the enhancement part is used.
    - locality properties:

      Spatial locality and temporal locality
  - Speed gap and the principles suggest memory hierarchy
- Design of cache memories
  - Placements or Mapping
    - Direct-mapped, set-associative mapped, and associative
  - Replacement algorithms and cache consistency

Memory System Architecture



Design of Memory Hierarchy

- Memory access time:

$T_{acc} = T_{hit} * hit\_ratio + T_{miss} * (1-hit\_ratio)$

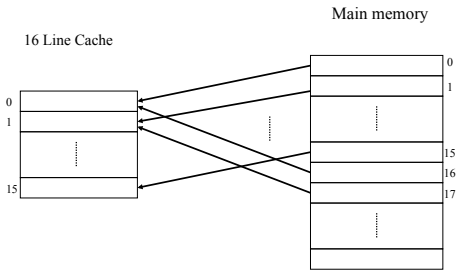
- Increase cache hit ratio
- Minimize miss penalty
- Design of cache memories
  - Placements or Mapping
  - Replacement algorithms
  - Write policy and cache consistency

Data Placement: Cache Mapping

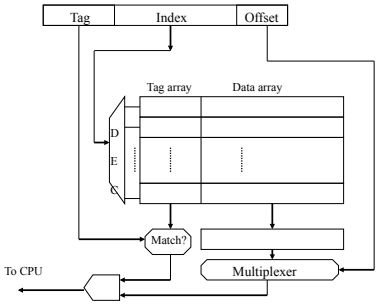
- Direct mapped cache:
  - Data with address A is mapped to exactly one cache location

    A modulo C, where C is number of lines in the cache
  - Simple logic, quick access: each address has 3 fields: Tag---Index---Offset
    - Offset gives a byte in a cache line; Index identifies the cache line corresponding to the data address, and tag compares with high order bits of the address to see if they match
  - Potential line interferences since many memory blocks map to a same cache line

Direct-Mapped Cache



Hardware Organization of Direct-Mapped



Fully Associative Mapped Cache

- Direct-mapped cache has high conflict misses
- Why not place a block at any free location?

→ Fully associative cache

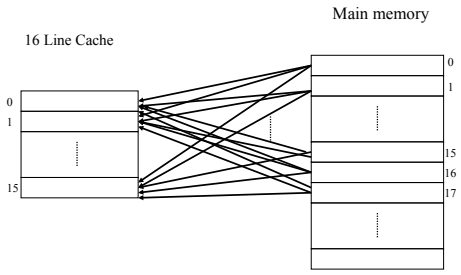
– No restriction as to where to place a cache line

– each address has 2 fields: Tag---Offset

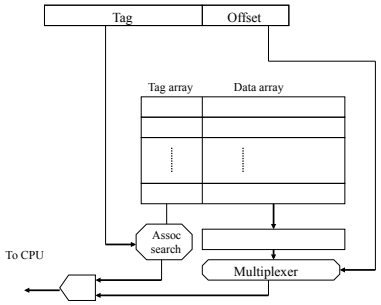
- Cache is accessed by associative search, matching tags: content addressable memory

– No line interferences, only capacity misses

Data Mapping of Fully Associative Cache



Hardware Organization of Fully Associative Cache



Set-Associative Mapped Cache

– A compromise/hybrid of the above two: associative map within a set and direct-map among sets

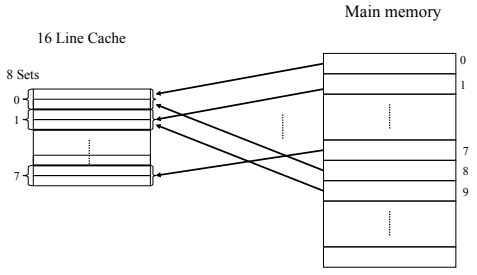
- Flexibility with a set to place data
- Simple indexing logic to identify a set
- d-way associative means d lines in a set
- Cache lines are divided into groups → sets

– Each address has 3 fields: Tag---Index---Offset

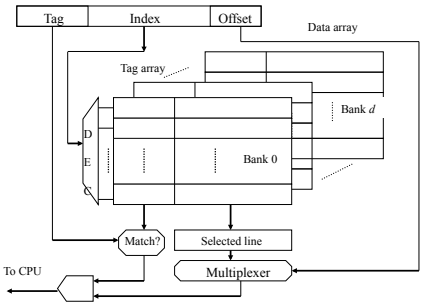
- Index here identify which set a line mapped to

– Reduce conflict misses and less complicated/faster than fully associative

Data Mapping of Set-Associative Cache



Hardware Organization of Set-Associative Cache



Replacement Algorithms

- LFU, LRU, Random, MRU, etc.
- LRU: Least Recently Used
  - An LRU counter is associated with each line
  - The LRU counters in a set form a logic stack
  - Bottom line is replaced

Memory Reference Sequence:

3	D
2	C
1	B
0	A

A B C D B

A is the LR item to be replace when CPU accesses E

LRU Counter Implementation

- Cache hit
  - The LRU counter of the referenced data is set to maximum
  - All other counters that are greater than the original counter of the referenced data are decremented by 1
- Cache miss
  - The line with counter 0 is replaced if the set is full
  - All the counters of lines in the set are decremented by 1
  - The counter of the new line is set to maximum

Write Policy

- Write-through
  - Every write updates both cached copy and memory copy
  - Write-through guarantee consistency but suffer from slow writes
- Write back
  - Write operations performed in cache only
  - Main memory updated only when changed line is replaced
  - Write-back: good performance but potential inconsistency